EXPERIENCE WITH FORM AL METHODS 'I'EC} INIQUES AT THE JET PROPULSION LABORATORY FROM A QUALITY ASSURANCE PERSPECTIVE

John C. Kelly, Ph.D., and Rick Covington, Ph.D.

Software Product Assurance Section
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, California 91109-8099

Internet: jkelly@spa1.jpl.nasa.gov, cov@spa1.jpl.nasa.gov

Abstract

Recent experience with Formal Methods (FM) in the Software Quality Assurance Section at the Jet Propulsion 1 ,ab is presented. An integrated Formal Method process is presented to show how related existing requirements analysis and FM techniques complement one another. Example application Of FM techniques such as formal specifications and specification animators arc presented. The authors suggest that the quality assurance organization is a natural home for the Formal Methods specialist, whose expertise can then be used to best advantage across a range of projects.

1. introduction

Recent studies of critical software subsystems have provided data which expose the software requirements quality problem facing current and future projects. The early stages of the software life cycle are especially prone to errors which can have a lasting influence on the reliability, cost, and safety of a system.

I highlights of these studies include the following:

- The highest density of major defects found through the use of software inspections was during the requirements phase (an average of 1 major defect found per 3 pages of requirements documentation). This was 7 times higher than the density of major defects found in code inspections
- Most hazardous software safety errors found during system integration and test of two unmanned spacecraft were the result of requirements discrepancies or interface specifications.
- Requirements errors are between 10 and 100 times more costly to fix at later phases of the software life cycle than at the requirements phase itself [1,2,6].
- One study found that the most frequent (30%) of all errors could be attributed to the faulty statement or understanding of requirements and specification while another found that early life cycle errors are the most likely to lead to catastrophic failures[7].

The.w problems indicate the need to advance the state-of-the-practice in the area of software requirements engineering. The later life cycle phases of detailed design and code are already supported by well defined methods, languages, and tools. However, current requirements engineering practices are mired in the use of ad hoc methods, ambiguous natural language specifications, and little or no automated support.

for example, arc the best candidates for the most rigorous application of Formal Methods. Figure 1 shows level of formality integrated with the Software Engineering Institute (SEI) Maturity Model [5,9]. The base and left-hand branch in Figure 1 shows the five levels of SEI's Process Maturity Model, which indicates an organization's capability to produce quality software products. A few NASA contractors have been rated in the 3-5 range of SEI's model (3 to 5 in Figure 1). For these organizations, a more formal approach could be used to break through their current software quality ceiling. The right-hand branch of Figure 1 (levels 4b to 7b) shows increasing levels of formality which can be applied to critical software projects in either forward or reverse engineering situations. Structure Modeling (4b) refers to using graphical models to determine the inter-relationship between requirements; Formal Specifications (5b) is the use of formal languages and associated tools to state and check requirements; specification animation (6b) enables the formal specification to be viewed dynamically; proofs (7b) can demonstrate whether the specifications arc sufficient to ensure critical properties.

The fraction of the application to be subjected to FM is also a function of application criticality and available resources. Finally, the life cycle phase of application development in which Formal Methods is applied determines which FM techniques are appropriate. For example, requirements analysis uses different tools from code verification. Note that life cycle is used in a broad sense, and that FM is not restricted to any specific life cycle model.

While the most rigorous assurance of correctness of specification comes from applying FM at the highest level of formality (i.e., proof of claims or properties concerning the specification), significant benefit to the development process can be attained from less formal aspects of the methodology. First, simply de-scribing a system or a critical subpart in a formal notation introduces a more rigorous way of thinking about the system. Second, type checking in a formal specification language subjects a specification to more rigorous scrutiny than merely checking that all expressions are of the correct type. In fact, type checking may give rise to type correctness conditions, or logical assertions about the declaration and usage of types in the specification which must be proven true before the specificat ion can be considered type-correct. Finally, specification animation, a technique for developing an executable program that exhibits the high-level behavior implied by the formal specification is often an effective informal alternative to proof for exploring the dynamic behavior of a specification.

III. Integrated Formal Methods Process Model

The authors' parent section (Software Product Assurance) is responsible for implementing product assurance programs on software subsystems at the Jet Propulsion Laboratory. Its role is to provide services and techniques to effectively mitigate risk during the development and maintenance of software subsystems. Within this section the authors are members of the Applied Research Group which evaluates new techniques and tools which support the objectives of improved software quality, then provide technology transfer courses and materials to allow projects to easily adopt the improved techniques. Currently Formal Methods and Object Oriented Design are key areas of activities. This section discusses how Formal Methods can be integrated into an ongoing software development process. The concurrent engineering

role of the Software Quality Assurance organization a natural organizational home for these techniques.

Figure 2 shows a simplified picture of how software requirements arc traditionally developed, The principal products developed include a textual description of the requirements, This usually includes English "shall" statements that describe high-level requirements or obligations the resulting software must satisfy. in addition, some projects usc various diagraming techniques to illustrate the structure of the requirements and to show their inter-relationship. These can include data flow diagrams, entity-relationships diagrams, state charts, and object diagrams. While these provide insight into the inter-relationships of requirements, they are usually a secondary consideration at this phase. High Level Test Plans arc developed during this phase to provide guidance for subsystem-level tests. Product assurance activities at this phase that have proven to be effective include Fagan-style inspections of both the textual requirements and the High Level Test Plan. These inspections arc conducted on segments of these two documents as they are being developed[6]. After the development of these documents (or significant segments of these documents in an incremental build life cycle), a baseline review is conducted prior to approval for the beginning of lower level engineering product development. While this process has been effective for many projects, there arc still significant opportunities for improvement. Current quality assurance approaches have the following limitations:

- These techniques arc mostly manual, thus making the finding of errors highly dependent upon the skill and diligence of the inspector and review teams.
- Even though a very high number of defects are found using these techniques, their prevalence and density indicate that there exist many errors which remain undiscovered.
- Some NASA projects have employed this approach to reach a quality ceiling on critical software subsystems. Therefore innovations are needed to push the project on toward new quality goals.

Figure 3 shows additional techniques that the authors and their associates have been investigating through demonstrations on critical NASA software subsystems. These additional interrelated techniq Lies include Structure modeling (in particular, the approach described by the Object Oriented Modeling (OMT)[10]) and formal specification (e.g., HOL[4] and PVS[11]). The quality assurance aspect of these engineering products are indicated by the attached gray boxes.

Structure modeling provides a graphical view of the inter-relationships between req uirements and, in the case of OMT models, the requ i red object model, dynamic model, and functional model. The graphical depiction of the desired system is easily reviewable through Fagan-style inspections. These inspections include project team members, who have a vested interest in the engineering product under review. Structure models can be used as a starting point for both specification animation and formal specifications. The arrows leading in from the edges indicate that, depending on the criticality of the subsystem, the current knowledge of requirements, and the resource of the project, any one of the three techniques potentially can be developed after the textual description. Creation of a formal specification requires a

translation of the requirements into a rigorous logical model supported by a formal specification language. When this technique is used, the formal specification becomes one of the primary engineering products of the requirements phase. The expected benefits from moving to a formal specification include 1) a reduction in ambiguity in the formal specification as compared with the textual description, 2) the variety and formal logical rigor of the quality assurance techniques which can be applied to the formal specification, and 3) the ability of the formal specification to significantly reduce the scope of the test plan without compromising its quality. Formal specifications support such automated analysis techniques as consistency checkers, steerable theorem provers, and proof checkers. Peer review inspections are still needed to verify that the formal specification faithfully represents the system that was intended.

Formal specifications require training and skills beyond most software engineering techniques. For this reason, we plan to use specialized FM teams to provide the development and checking of formal specifications, and to make this expertise available to project developers. in such an arrangement, only a reading knowledge of formal specification languages is required for most of the software development team. Furthermore, specification animation provides insight into the requirements which is not possible through structured modeling or formal specification alone. Specification animation allows the requirements to be viewed dynamically by supplying a program that demonstrates the execution of the high level functionality of the subsystem under study. Specification animations also provide a reasonableness check on the formal specification by providing the project team with an executable representation of the requirements. This animation serves as a high-level prototype which is built to confirm the description of the subsystem, rather than as a low-level demonstration of the feasibility of a particular feature. The authors have found specification animation to be very useful in both forward as well as reverse engineering situations, and details of example usage is given in the next section.

IV. Case Studies

in this section, we present a case study of how FM was used in a forward engineering scenario for the study of the floating point math for the MIL-STD-1750A microprocessor, and in a reverse engineering sense for a generic spacecraft guidance, navigation and control (GNC) function.

A. MIL-STD-1750A Floating Point

This section reports on some of the results from a study of the floating point arithmetic of the Mll,-Sl'I)-1750A microprocessor[12]. The purpose of the study was to generate a formal specification of the definition of the floating point operations from the MIL-STD document, and to then explore the formal specification for internal logical consistency and ambiguity which could lead to incorrect implementations of the standard in silicon. The specification language used was HOL (Higher-Order Logic), a public domain tool maintained by Cambridge University (UK) 4.

Consideration of addition, multiplication, and division found no serious problems, but the definition of subtraction was found to be ambiguous. The definition from the MIL-STD for floating point subtraction is as follows:

> The floating point Derived Operand, DO, is floating point subtracted from the

contents of registers RA and RA+1. The result is stored in registers RA and RA+l,

'I"he process of this operation is defined as follows: the mantissa of the number with the smaller algebraic exponent is shifted right and the exponent incremented by one for each bit shifted until the exponents are equal. The mantissa of the DO is then subtracted from (RA,RA+1). If the difference overflows the 24-bit mantissa, then it is shifted right one position, the sign bit restored, and the exponent incremented by one. If the exponent exceeds 7F hex as a result of this increment operation, overflow occurs and the operation is terminated. If the sum does not result in exponent overflow, the result is normalized. If during the normalization process the exponent is decremented below 80 hex, then underflow occurs and a zero is inserted for the result.

The essentials of the resulting HOL specification for 1750A single precision floating point subtraction are contained in the following HOL definitions.

```
let float_axiom =
define_type 'float_axiom'
'float = FLOAT word24
word8' ; ;

let spf_sub =
new_definition
 (' spf_sub_def`,
 "spf_sub (a :fl oat)
          (b :float)  =

 let. argnorm =
    ((spf_isnormalized a)
    /\ (spf_isnormalized
b)) in

 Jet ai = (spf_to_ispf a)
in
 Jet bi = (spf_to_ispf b)
in
 let   bitc =
    (ispf_twoscomp bi) in
 let azero =
    (spf_equals a
spf_zero) in
 let bzero =
    (spf_ equals b
spf zero) in

 ((~argnorm) =>
    (SPFREC ARB F F  '1'  F)|
 ((bzero) =>
    (SPFREC a F F F  E") |
 (ispf_do_acid ai
bi_tc)))");;
```

The first definition establishes the type "flea t" as a record composed of a 24 bit mantissa and an 8 bit exponent. The second defines the single precision floating point subtraction operation "spf_sub". The series of six "let" statements within "spf_sub" establish the value of several intermediate results. To interpret these statements, note that HOL expressions follow a lisp-like syntax. A parenthesized list of symbols indicates a function call, with the first element in the list the name of the function, and the remaining symbols the arguments to the function. "argnorm" is a Boolean indicating whether or not the original arguments "a" and "b" are normalized. "ai" and "hi" represent extended precision arithmetic versions of the original arguments''a''and "b". "bi_tc" represents the two's complement of "hi" in extended arithmetic. "azero" and "bzero" arc Booleans which indicate whether or not "a" and "b" respectively arc equal to zero. The final three lines of "spf_sub" implement an "if-then-else" which constructs the return value for the function. If arguments arc not normalized, the result is undefined, else if "b" is equal to zero, "a" is returned, else the value of "a" added to the two's complement of "b" is returned.

The final step in the definition of "spf_sub" implicitly assumes that "a" added to the negative (two's complement) of "b" produces the same value as "b" directly subtracted from "a". However, counterexamples can be found with some pencil-and-paper analysis. An examination of the highlighted passages above from the MIL-STD document reveals that the standard is incomplete or ambiguous on the specific subtraction algorithm to use. Since two of the known possible choices -- -"add-the-negative" and "direct subtraction" -- can be shown to give different results for some pairs of identical operands, the possibility exists that this ambiguity could be propagated into distinct implementations of the standard in silicon. Although most fabricators are aware of this ambiguity and have resolved it in a sensible fashion, the exercise demonstrated how quickly fundamental questions about completeness and consistency of a specification can be grasped once the specification is stated formally. In this study, most of the interesting conclusions were obtained by subjecting the problem only to a moderate level of formalism (i.e., type checking and specification animation, but no proofs).

B. Spacecraft GNC

This section presents the results of a reverse-engineering study of a generic spacecraft GNC (guidance, navigation, and control) subsystem. The goal of the study was to derive requirements for a system supported only by implementation-level external documentation. From a quality assurance perspective, the goals of the study were:

- to understand what were the important global properties (i.e., the true requirements) which were obscured by implementation detail
- to produce support products that would assist in the task of bringing personnel new to the project up to speed
- to produce support products that would assist in the task of determining global effects of modifications (maintenance)
- to provide an alternative to the system's current maintenance approach which uses heavy manual analysis

The problem statement for the GNC function is, given a commanded direction along three rotation axes (roll, pitch, and yaw), determine the best combination of jets to fire to satisfy the command. Jets are distributed about the body of the spacecraft and fire in fixed directions. A table of precomputed values which describe the expected rotational effects from individual jet firings is available to the algorithm.

An example of the documentation for one subfunction is shown in figure 4. This function computes four table lookup indices (Index 1 through Index 4) into a table of jet rotation characteristics as a function of the input GNC system mode, represented by the four inputs, mode 1 through mode 4, and predefined integer constants 1'1 through 1'8. Dotted lines serve as switch controls, causing in effect the slanted lines to be "thrown" to the switch terminal that matches the value of the switch control. The figure defines its function with precision, but not with clarity. Most importantly, there is no structure to the figure that suggests any rationale for why the function works the way it does.

In the absence of other information, the best strategy for beginning the process of discovering the lost rationale of such functions was to write a specification animator to mimic the function's behavior according to the available documentation. This

strategy was straightforward to implement for the function in question since the ranges for all inputs were discrete and relatively small, making exhaustive testing feasible. From studying the values of the output indices, a more obvious structure quickly became apparent. This new structure is presented in flowchart form in figure 5, and in tabular form in figure 6. It was shown by exhaustive testing to be equivalent to the original documentation (figure 4). From figures 5 or 6, the underlying logic of the function becomes clearer. First, the actual system modes can be recognized from the values of the inputs, Mode 1 through Mode 4. The flowchart implies that there are two major modes (Mode 4 = on and Mode 4 = off, which we will refer to as "4 on" and "4 off"). Mode "4 on" is further broken down into 3 submodes, "3 Oil", "3 off/2 on", and "3 off/2 off", while major mode "4 off" has 110 submodes. Finally, the flowchart reveals that assignment of values to indices 1 and ? is decoupled from the assignment of values to Indices 3 and 4.

This example illustrates how in a reverse engineering task, specification animators motivate the discovery of true underlying structure which may not be readily apparent from lower-level detail. If the exact needs of the task do not demand the highest levels of formal logical rigor, the flowchart or table could be a sufficient final product. If more formalism is required, the flowchart or table could then be translated into a formal specification language, where it could be subjected to automated deductive analysis. In the latter case, note that specification animators still play an important intermediate role. It is possible to translate the original figure (figure 4) directly into a formal specification language, skipping the specification animation stage, but doing so misses the opportunity to reexpress the function specification to expose the logical essence of what the function does. The automated logical analysis of the specification has a much better chance of producing useful results if the specification itself elucidates rather than obscures this essence.

v. conclusions

Based on these pilots, the JPL experience supports the view that FM is ready for critical software applications in space applications. We have described concrete process steps which were evolved in the course of the JPL pilot studies to allow FM to be introduced into existing development and quality assurance processes, without the need to significantly increase up-front costs nor to extensively train large groups of software engineers. We have also investigated the expansion of the FM idea to include high-level requirements simulators or "requirements animators" and have proposed the integration of formal methods with object oriented diagrams. The use of animators facilitates a process in which quality assurance personnel provide FM expertise and development personnel provide application domain expertise. Animators then become a useful medium for communication of results of FM analysis between personnel who need the results but who need not be experts in FM. However, our experience with the link between Object Structure Modeling and Formal Methods is far more preliminary and we can not confirm or reject the usefulness of this addition at the requirement phase at this time. This experience suggests that quality assurance organizations could be a natural home for FM in many high reliability systems development environments.

Acknowledgments

The research described in this paper was carried out by the Jet Propulsion

Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Partial funding was provided by an 1{'1'01' from **NASA** Code **Q** for a multicenter investigation into Formal Methods at the Jet Propulsion Laboratory, Johnson **Space** Center, and **1** angley Research Center.

References

1 Basili, V. I{. and Perricone, B.T. "Software Errors and Complexity: **A n** Empirical Investi gation", *Communications of the ACM*, ?1(1): 42-52, January, 1984.

2 Boehm, B.W. "Software Engineering Economics", *IEEE Transactions on Software Engineering*, SE-10(1): 4-21, January, 1984.

3 Covington, R.G., Abernethy, K., and Cullyer, J. C., "Using For Ins] Methods to Model the 1750A Microprocessor Floating Point Arithmetic", submitted for publication.

4 Gordon, M.J.C., "1 101,: A Proof Generating System for Higher Order Logic", in *VLSI Specification, Verification and Synthesis*, G. Birtwistle and P.A. Subrahmanyam, eds., 73-]?8, Kluwer Academic Publishers, 1988.

5 Humphrey, W.S., "Characterizing the Software Process: A Maturity Framework", CMU/SEI-87-TR-11, Software Engineering Institute, June 1987.

6 Kelly, J.C'., Sherif, J. S., and Hops, J. "An Analysis of Defect Densities Found During Software Inspections", *Journal of* Systems *and Software*, vol 17, 111-117, January, 1992.

7 Leveson, N.G. "Software Safety: Why, What, and How", *A CM Computing Surveys*, 18(2): 125-163, June, 1986.

8 Lutz, R.R. "Analyzing Software Requirements Errors in Safety - Critical Embedded Systems", to appear in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA, January, 1993.

9 Paulk, M.C., et al., "Capability Maturity Model for Software", CMU/SEI-91-TR-24, The Software Engineering Institute, August,]99],

10 Rumbaugh, J., Blaha, M., Premerlani, F.E., and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

11 Shankar, N., Owre, S., and Rushby, J.M., "The PVS Specification Language (Beta Release)", SRI International, Menlo Park, CA, March 31, 1993.

1? US Department of Defense, "Military Standard Sixteen-bit Computer Instruction Set Architecture", MIL-STD-1750A (USAF), July 2, 1980.

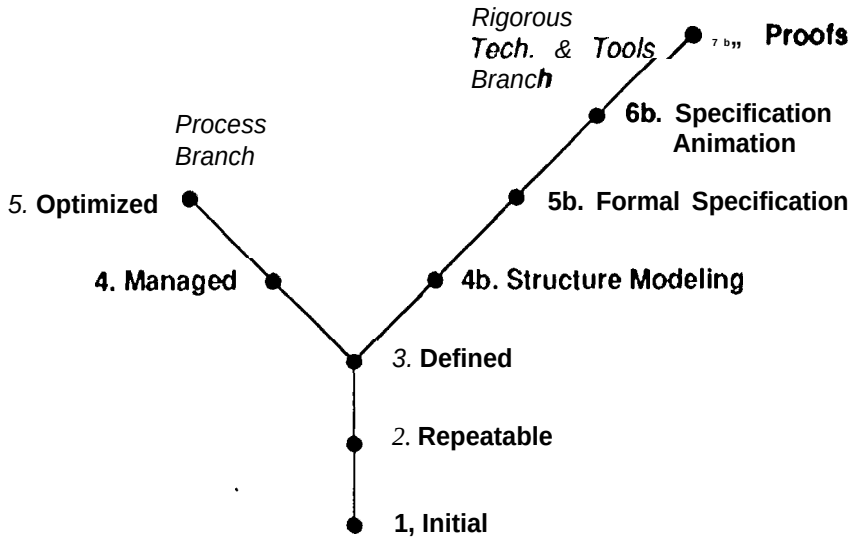


Figure 1.: Levels of Expected Quality

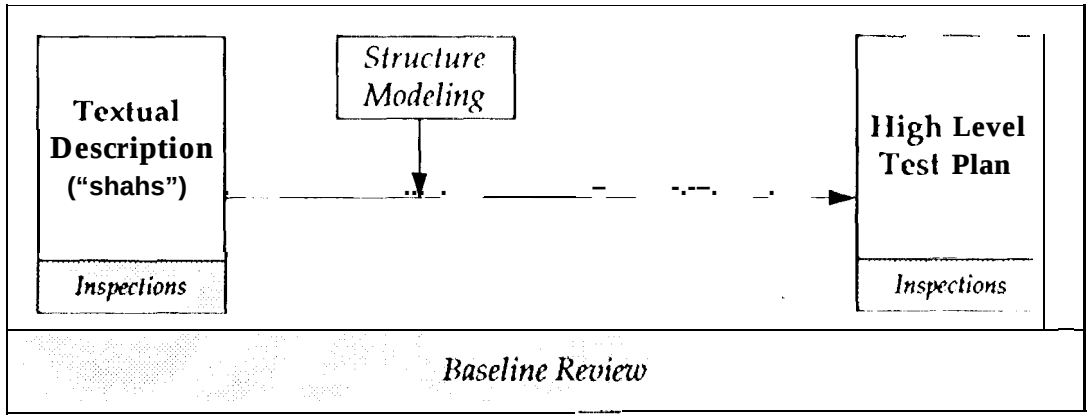
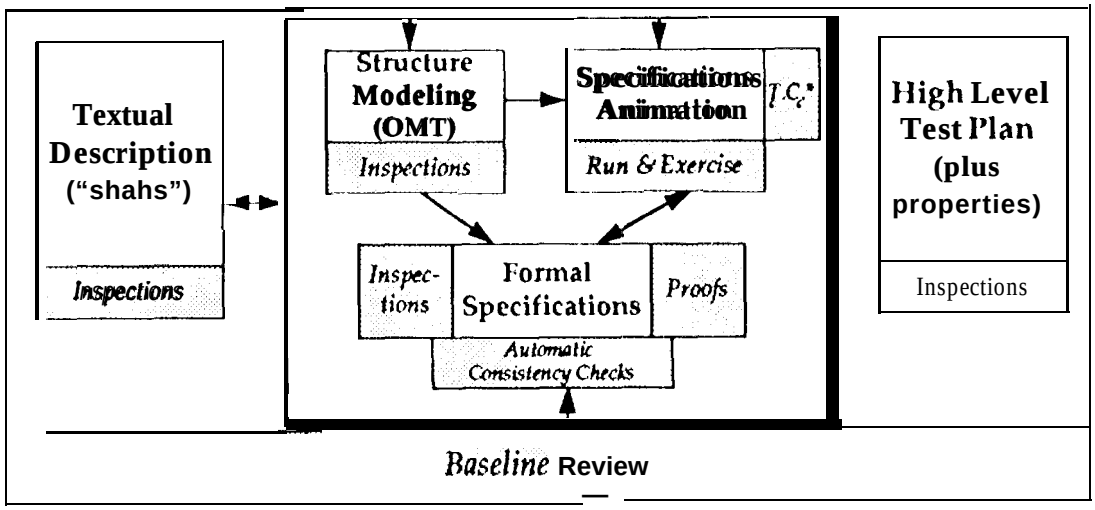


Figure 2.: Traditional Requirements Development and Quality Assurance



*T.C. = trace to formal specification check

Figure 3.: Integrated Formal Methods Process Model